»Лекционен курс

»ООП1 (Java)

Контрол на достъпа ›

Регистрация

Скриване на информация

Мотивация

- Какво предимство може да има скриването на информация?
- В компютърните науки скриването на определена информация се счита за добра техника за програмиране, която:
  - Улеснява работата на програмиста;
  - По-лесно разбираем код.
- Това е начин да се избегне „претоварване с информация“

Мотивация

- Програмист, който използва метод от съществуващ клас не трябва да знае подробностите за кода в тялото на дефиницията на метода, за да може да използва метода.
- Ако методът или друга част от софтуера е написан добре, програмист, който използва метода, трябва да знае само какво прави методът, а не как го прави.
- Проблемът не е, че кодът съдържа някаква тайна, която е забранена за програмиста.
- Въпросът е, че преглеждането на кода няма да помогне да бъде използван метода.
- Обратно, ще натовари програмиста с излишни детайли, които го отвличат от задачите му.

Скриване на информация

- Скриване на информация: проектиране на методи, така че да могат да се използват без необходимост от разбиране детайли на кода.
- Терминът подчертава факта, че програмистът действа така, сякаш тялото на метода е скрито от него.

Скриване на информация

» Абстракция (капсулиране):

> В този контекст двата термина означават едно и също нещо.

> Използване на термина абстракция не би трябвало да е изненадващо – когато се абстрахираме от нещо, губим част от детайлите.

> Напр., резюме на статия или книга е кратко описание на съдържанието, за разлика от целите текстове.

Модификатор public

Модификатор public

Модификаторът public, когато е приложен към:

- Клас
- Метод
- Променлива на обект

означава, че всеки друг клас може директно да използва или да осъществи достъп до класа, метода или променливата по име.

```java
import java.util.Scanner;
public class SpeciesSecondTry {
    public String name;
    public int population;
    public double growthRate;
    public void readInput()    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Име на вида?");
        name = keyboard.nextLine();
        System.out.println("Популация на вида?");
        population = keyboard.nextInt();
        System.out.println("Въведи индекс на нарастване " + "(% увеличение на година):");
        growthRate = keyboard.nextDouble();
    }
    public void writeOutput() {
        System.out.println("Име = " + name);
        System.out.println("Популация = " + population);
        System.out.println("Индекс нарастване = " + growthRate + "%");
    }
    public int predictPopulation(int years) {
        int result = 0;
        double populationAmount = population;
        int count = years;
        while ((count > 0) && (populationAmount > 0)) {
            populationAmount = (populationAmount +
                    (growthRate / 100) * populationAmount);
            count--;
        }
        if (populationAmount > 0)
            result = (int)populationAmount;
        return result;
    }
}
```

модификатор public – променливите на обект директно достъпни по име от друг клас.

Не добра практика – противоречи на принципите на ООП.

```
import java.util.Scanner;
public class SpeciesSecondTry {
    public String name;
    public int population;
    public double growthRate;
    public void readInput()   {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Име на вида?");
        name = keyboard.nextLine();
        System.out.println("Популация на вида?");
        popula
        System
        growthR
    }
    public voi
        System
        System
        System
    }
    public int
        int re
        double
        int co
        while
            po
                    (growthRate / 100) * populationAmount);
            count--;
        }
        if (populationAmount > 0)
            result = (int)populationAmount;
        return result;
    }
}
```

- Въпреки че е нормално да имаме публични класове и методи, не е добра практика променливите на обекти да са публични.
- Обикновено всички променливи на обектите трябва да са частни.
- Използваме модификатора private.
- Ключовите думи public и private са примери за модификатори на достъп.

Пример: използване на класа SpeciesSecondTry

```
import java.util.Scanner;
public class SpeciesSecondTryDemo {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        SpeciesSecondTry speciesOfTheMonth = new SpeciesSecondTry();
        System.out.println("Въведи данни за вида:");
        speciesOfTheMonth.readInput();
        speciesOfTheMonth.writeOutput();
        System.out.println("Въведи годините за прогноза:");
        int projectedYears = keyboard.nextInt();
        int futurePopulation = mySpecies.
                predictPopulation(projectedYears);
        System.out.println("След " + projectedYears + " години, ");
        System.out.println("популацията ще бъде "+ futurePopulation);
         speciesOfTheMonth.name = "Родопско лале";
         speciesOfTheMonth.population = 10;
         speciesOfTheMonth.growthRate = 15;
         System.out.println("The new Species of the Month:");
         speciesOfTheMonth.writeOutput();
         System.out.println("In ten years the population will be " +
         speciesOfTheMonth.predictPopulation(10));
             }
         }
```

```
import java.util.Scanner;
public class SpeciesSecondTry {
    private String name; // private!
    public int population;
    public double growthRate;
    public void readInput()   {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Име на вида?");
        name = keyboard.nextLine();
        System.out.println("Популация на вида?");
        population = keyboard.nextInt();
        System.out.println("Въведи индекс на нарастване " + "(% увеличение на година):");
        growthRate = keyboard.nextDouble();
    }
    public void writeOutput() {
        System.out.println("Име = " + name);
        System.out.println("Популация = " + population);
        System.out.println("Индекс нарастване = " + growthRate + "%");
    }
    public int predictPopulation(int years) {
        int result = 0;
        double populationAmount = population;
        int count = years;
        while ((count > 0) && (populationAmount > 0)) {
            populationAmount = (populationAmount +
                    (growthRate / 100) * populationAmount);
            count--;
        }
        if (populationAmount > 0)
            result = (int)populationAmount;
        return result;
    }
}
```

модификатор private за name

Какъв ефект?

Пример

```
import java.util.Scanner;
public class SpeciesSecondTryDemo {
    public static void main(String[] args) {
        Scanner keyboard = new Scanner(System.in);
        SpeciesSecondTry speciesOfTheMonth = new SpeciesSecondTry();
        System.out.println("Въведи данни за вида:");
        speciesOfTheMonth.readInput();
        speciesOfTheMonth.writeOutput();
        System.out.println("Въведи годините за прогноза:");
        int projectedYears = keyboard.nextInt();
        int futurePopulation = mySpecies.
                predictPopulation(projectedYears);
        System.out.println("След " + projectedYears + " години, ");
        System.out.println("популацията ще бъде "+ futurePopulation);
         speciesOfTheMonth.name = „Родопско лале";
         speciesOfTheMonth.population = 10;
         speciesOfTheMonth.growthRate = 15;
         System.out.println("The new Species of the Month:");
         speciesOfTheMonth.writeOutput();
         System.out.println("In ten years the population will be " +
         speciesOfTheMonth.predictPopulation(10));
             }
         }
```

Невалиден достъп

Частни променливи и методи

Частни променливи

- Когато променлива на обект е частна, името ѝ не е достъпно извън дефиницията на класа.
- Можем да използвате името ѝ по какъвто и да е начин в който и да е метод в дефиницията на класа.
- По-специално, можем директно да променим стойността на променливата.
- Извън дефиницията на класа обаче не можем да имаме пряка референция към името на променливата.

```java
import java.util.Scanner;
public class SpeciesThirdTry {
    private String name;
    private int population;
    private double growthRate;
    public void readInput()    {
        Scanner keyboard = new Scanner(System.in);
        System.out.println("Име на вида?");
        name = keyboard.nextLine();
        System.out.println("Популация на вида?");
        population = keyboard.nextInt();
        System.out.println("Въведи индекс на нарастване " + "(% увеличение на година):");
        growthRate = keyboard.nextDouble();
    }
    public void writeOutput() {
        System.out.println("Име = " + name);
        System.out.println("Популация = " + population);
        System.out.println("Индекс нарастване = " + growthRate + "%");
    }
    public int predictPopulation(int years) {
        int result = 0;
        double populationAmount = population;
        int count = years;
        while ((count > 0) && (populationAmount > 0)) {
            populationAmount = (populationAmount +
                    (growthRate / 100) * populationAmount);
            count--;
        }
        if (populationAmount > 0)
            result = (int)populationAmount;
        return result;
    }
}
```

модификатор private – променливите на обект достъпни по име само в класа.

Класическо ООП.

Частни методи

- Методите могат да бъдат също частни.
- Ако методът е дефиниран като частен, той не може да бъде извикан извън дефиницията на класа.
- Частните методи могат да бъдат извиквани в рамките на дефиницията на всеки друг метод в същия клас.
- Повечето методи са публични, но ако имате метод, който ще се използва само в рамките на дефиницията на други методи от този клас, има смисъл да направим този метод „помощен“ (частен).
- Използването на частни методи е друг начин за скриване на детайлите в класа.

Само за сведение: класовете могат също да бъдат private

Обобщение

- Трябва да направим всички променливи на обект от клас частни.
- По този начин принуждаваме програмиста, който използва класа да осъществява достъп до променливите на обекта само чрез методите на класа.
- Това позволява на класа да контролира как програмиста гледа или променя променливите на екземпляра.
- Следващият пример илюстрира защо е важно да правим променливите на обектите частни.

Пример

```
/**
 Class that represents a rectangle.
 */
public class Rectangle {
    private int width;
    private int height;
    private int area;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

Пример

Какъв резултат?

```
Лицето на квадрата е 50

Process finished with exit code 0
```

```
public class RectangleDemo {
    public static void main(String[] args)  {
        Rectangle box = new Rectangle( );
        box.setDimensions(10, 5);
        System.out.println("Лицето на квадрата е " +
                box.getArea());
    }
}
```

Какво би станало ако ...?

... променливите в Rectangle бяха Публични.
След създаването на правоъгълник с 10 на 5 ще можем да променим стойността на която и да е или всички променливи на инстанцията.

```
/**
 Class that represents a rectangle.
 */
public class Rectangle {
    public int width;
    public int height;
    public int area;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

Така че, докато лицето на правоъгълника е 50, можем да промените ширината, например, на 6, като напишем ...

```
public class RectangleDemo {
    public static void main(String[] args)  {
        Rectangle box = new Rectangle( );
        box.setDimensions(10, 5);
        box.width = 6; // if width were public.
        System.out.println("Лицето на квадрата е "
                box.getArea());
    }
}
```

Правейки ги частни ограничаваме такъв достъп и промяна.

Пример

```
/**
 Another class that represents a rectangle.
 */
public class Rectangle2 {
    private int width;
    private int height;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
    }
    public int getArea() {
        return width * height;
    }
}
```

- Rectangle2 има абсолютно същите методи като предишния клас Rectangle, но ги прилага по малко по-различен начин.
- Новият клас изчислява лицето на правоъгълника само когато е извикан методът getArea.
- Освен това лицето не се записва в обектна променлива.

Пример

```
public class Rectangle2Demo {
    public static void main(String[] args)  {
            Rectangle2 box = new Rectangle2( );
            box.setDimensions(10, 5);
            System.out.println("Лицето на квадрата е " +
                    box.getArea());
    }
}
```

- Ще използваме класа Rectangle2 по същия начин както Rectangle.
- Единствената промяна - заменяме Rectangle с Rectangle2.

- Два класа (Rectangle и Rectangle2) се държат по един и същ начин.
- Тоест, това, което правят, е същото, но начина, по който изпълняват задачите си, се различава.

```
public class Rectangle {
    public int width;
    public int height;
    public int area;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

- Rectangle обаче има обектна променлива за лицето
- Методът setDimensions в Rectangle изчислява лицето и го съхранява в area.
- След това методът getArea просто връща стойността в областта на променливата на екземпляра.

```
public class Rectangle2 {
    private int width;
    private int height;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
    }
    public int getArea() {
        return width * height;
    }
}
```

В Rectangle2 методът getArea изчислява и след това връща лицето, без да го запазва.

Дали един от тези два класа е „по-добър“ от другия?

```
public class Rectangle {
    public int width;
    public int height;
    public int area;
    public void setDimensions(i
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

```
public class Rectangle2 {
    private int width;
    private int height;
    public void setDimensions(int newWidth, int newHeight) {
        width = newWidth;
        height = newHeight;
    }
    public int getArea() {
        return width * height;
    }
}
```

Отговорът зависи от това какво имаме предвид под „по-добър“ и как използваме класа.
Можем да направим следния извод:

- Rectangle използва повече памет, защото има допълнителна обектна променлива.
- Rectangle винаги изчислява лицето, дори и да не е необходимо.

Дали един от тези два класа е „по-добър“ от другия?

```
public class Rectangle {
    public int width;
    public int height;
    public int area;
    public void setDimensions(in
        width = newWidth;
        height = newHeight;
        area = width * height;
    }
    public int getArea() {
        return area;
    }
}
```

```
public class Rectangle2 {
    private int width;
    private int height;
    public void setDimensions(in
        width = newWidth;
        height = newHeight;
    }
    public int getArea() {
        return width * height;
    }
}
```

Вторият извод е:

- Използването на Rectangle може да изисква повече компютърно време - за да оценим това, трябва да си представим, че и двата класа имат още няколко метода, така че да не се извиква getArea е реална възможност.
- Ако рядко питаме за лицето на определен правоъгълник, използването на Rectangle2 спестява както времето за изпълнение, така и памет
- Ако обаче многократно извикваме getArea за определен правоъгълник, използването на Rectangle ще спести време за изпълнение, защото изчислява площта само веднъж.

Отново за капсулиране

Отново за капсулиране

- До този момент не се притеснявахме за контрола на достъпа, понеже Java предоставя достъп по подразбиране.
- За разгледаните примери членовете на даден клас са свободно достъпни за другия код в програмата.
- Макар че е удобно за прости класове, този вид достъп е недостатъчен за много реални ситуации.
- Тук представяме други възможности за контрол на достъпа в Java.

Отново за капсулиране

- В подкрепата на капсулирането класът предоставя две основни предимства:
  - Първо, свързва данните с кода, който ги манипулира.
  - Второ, осигурява средствата, чрез които достъпът до членовете може да бъде контролиран.
- Въпреки, че подходът на Java е малко по-сложен, по същество има два основни типа членове на класа:
  - public
  - private

Отново за капсулиране

- Публичен член на класа може да бъде свободно достъпен чрез код, дефиниран извън неговия клас.
- Частният член на класа може да бъде достъпен само чрез методи, определени от неговия клас:
  - Чрез използването на частни членове достъпът се контролира

Отново за капсулиране

- Ограничаването на достъпа до членовете на класа е основна част от обектно-ориентираното програмиране.
  - Подпомага предотвратяване злоупотреба с даден обект.
- Като разрешаваме достъп до private данни само чрез добре дефиниран набор от методи (интерфейс), можем да предотвратим присвояване на неправилни стойности на тези данни.
  - Напр., извършване проверка на допустимите стойности.

Отново за капсулиране

- Не е възможно код извън класа да присвои директно стойност на private член.
- Можем също да контролираме точно как и кога се употребяват данните в даден обект.
- Така, когато е коректно имплементиран, класът става "черна кутия", която може да бъде използвана.
  - Без намеса в вътрешните имплементации.

Класове и стандартни типове данни

- Това, което правим в ООП е създаване на нови типове данни.
- Едно число е също тип – има определени характеристики и поведение.
  - Разлика: в ООП дефинираме класове, подходящи за решаване на дадена задача, вместо да ни бъде наложено да използваме съществуващ тип данни, който е бил проектиран да представя единица в машината.
- Добавяйки нови типове данни, специфични за решаване на различни задачи, ние разширяваме езика за програмиране.

Доставяне на прости решения

- Използването на обектно-ориентирани техники може да редуцира голям брой задачи до едно просто решение:
  - След като веднъж е създаден един клас, можем да създаваме произволен брой обекти.
  - Тези обекти могат да се обработват като елементи от решението на дадена задача.
  - Едно от предизвикателствата на ООП е създаване на съответствие между пространството на задачата и обектите от пространството на решението.

Заявки към обекти

- Как можем да накараме един обект да прави полезни за нас неща?
  - Трябва да съществува начин за правене заявки към обектите.
- Всеки обект може да удовлетворява само определени заявки
  - Заявките, които можем да правим към един обект са дефинирани от неговия интерфейс.
  - Типът е този, който определя интерфейса.

Заявки към обекти

- Интерфейсът установява какви заявки можем да правим за определен обект.
  - Някъде обаче, трябва да съществува код, който да изпълнява заявката.
  - Това, заедно със скритите данни включва имплементацията.

Създатели и клиенти на класове

Създаване и използване на класове

- В ООП различаваме две основни роли:
- Създатели на класове
  - Целта им е създаване на нови класове, като разкриват само това, което е необходимо на клиентите и пази всичко останало.
  - Скритите части не могат да се използват от клиентите-програмисти.
  - Т.е., създателите могат да ги променят без да се тревожат за въздействието върху другите.
- Клиенти на класове
  - Целта на клиент-програмистите е да създава сбирка от инструменти (класове), които да използва за бързо разработване на приложения.

Права на клиентите

- Да декларират променливи от тип class
- Да създават обекти на класа, използвайки конструктури
- Да изпращат съобщения към обектите, използвайки методите на инстанции, дефинирани в класа
- Да познават публичния интерфейс на класа
  - Имената на методите на инстанции, броя и типа на параметрите, типа на резултатите
- Да знаят кои методи на инстанции променят обектите

Права на създателите на класове

- Да дефинират публичния интерфейс на класа
- Да скриват от клиентите всички детайли на имплементацията
- Да защитават “вътрешните” данни от достъп на клиентите
- Да променят детайли на имплементацията по всяко време запазвайки публичния интерфейс
  - Ако се налага промена на интерфейса - съгласувано с клиентите

Контрол на достъп

- Взаимоотношенията между създателите и клиентите трябва да бъдат регулирани.
- Ако всички елементи на един клас са достъпни за всички, тогава клиентите могат да правят всичко с класа и не съществува начин за налагане на правила.

Контрол на достъп

- За целта съществува контрол на достъп
  - Да се държат клиентите далече от частите, които не трябва да се пипат.
    - Вътрешната обработка на данните, която не е част от интерфейса.
    - В действителност, това е услуга за потребителите – лесно могат да се ориентират кое е важно за тях и кое могат да игнорират.
  - Да се позволява на създателите да променят вътрешната структура на класа, без това да влияе на клиентите.
- Java използва определени ключови думи за установяване границите на един клас.
-

Спецификатори за достъп

- public
  - Следващите спецификации достъпни за всички.
- private
  - Никой, освен създателят няма достъп до тези дефиниции в рамките на действието на спецификатора.
  - Оперира като стена между създателя и клиента.
  - При опит за установяване на достъп – грешка по време на компилиране.
- protected
  - Като private, с изключение на това, че наследяващият клас има достъп.
- „приятелски“ достъп (по подразбиране)
  - Ако не се използва някой от горепосочените.
  - Достъп в същия пакет.

Капсулиране

- Капсулирането включва скриване на подробностите за това как работи даден софтуер.
- Сега, след като имаме повече познания за Java и скриване на информация, можем да разширим тази основна дефиниция - капсулирането е процес на скриване на всички детайли на дефиниция на клас, които не са необходими за да се разбере как се използват обектите от класа.

Капсулиране

- За да използва софтуер, програмистът не се нуждае от всички подробностите за неговата дефиниция.
- По същия начин част от софтуера трябва да бъде капсулирана така, че да се виждат само необходимите контроли и да се скрият неговите детайли.
- По този начин програмистът, който използва софтуера, е пощаден от ненужни детайли.

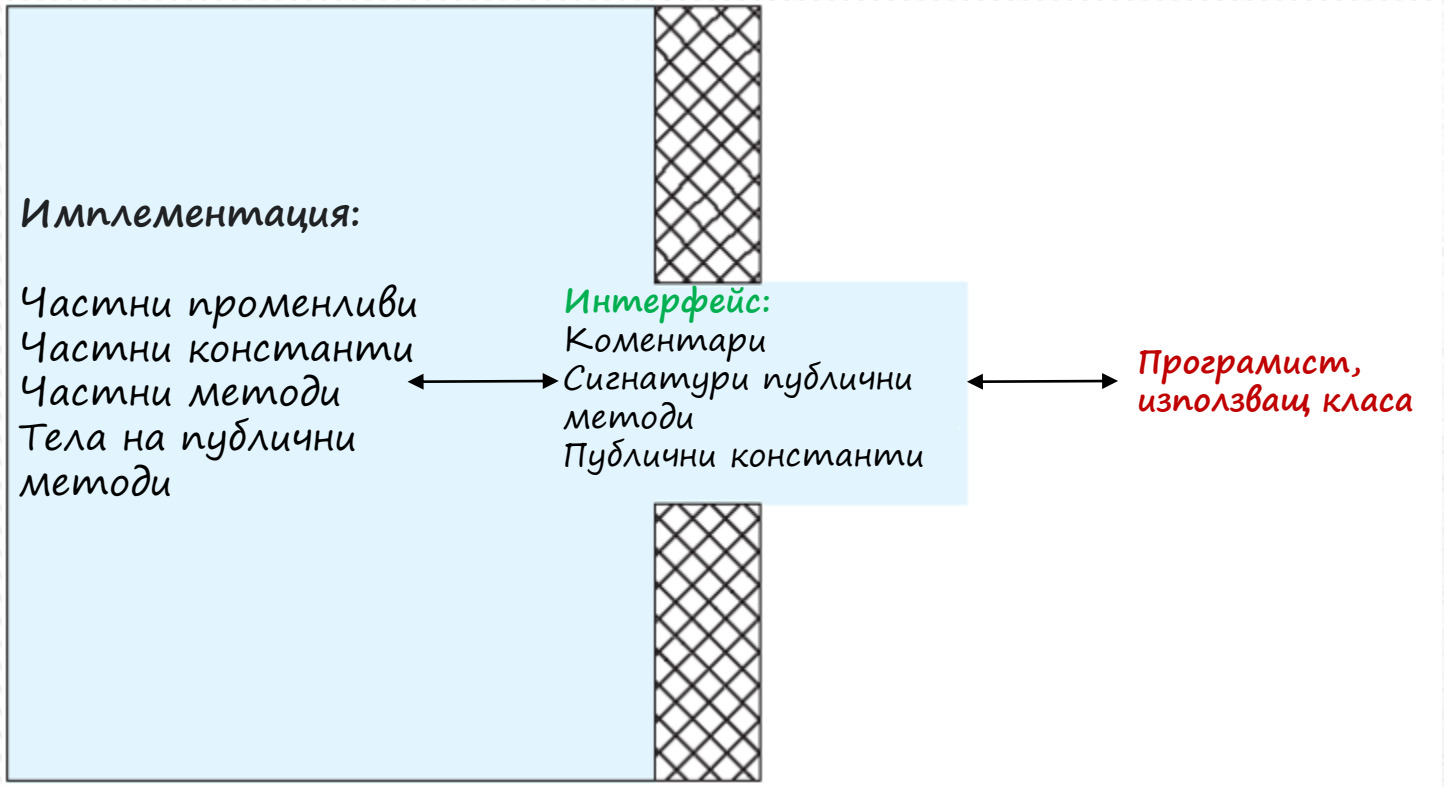
Капсулиране

- Имплементирането на клас се състои от всички частни елементи на дефиницията на класа, главно променливите на частния екземпляр на класа, заедно с дефинициите както на публичните методи, така и на частните методи.
- Интерфейсът на класа и имплементация не са отделни в Java кода, а се смесват заедно.
- Въпреки че имаме нужда от имплементация на клас, когато използваме класа, не трябва да знаем нищо за имплементация за да напишем нашата програма

Капсулиране

- Когато дефинираме клас, използвайки принципа на капсулиране, трябва да отделим концептуално интерфейса на класа от имплементацията, така че интерфейсът да е опростено и безопасно описание на класа.
- Един от начините да мислим за това разделяне е да си представим стена между изпълнението и интерфейса, с добре регулирана комуникация през стената.

Дефиниция на клас - обобщение



Добра практика за капсулиране

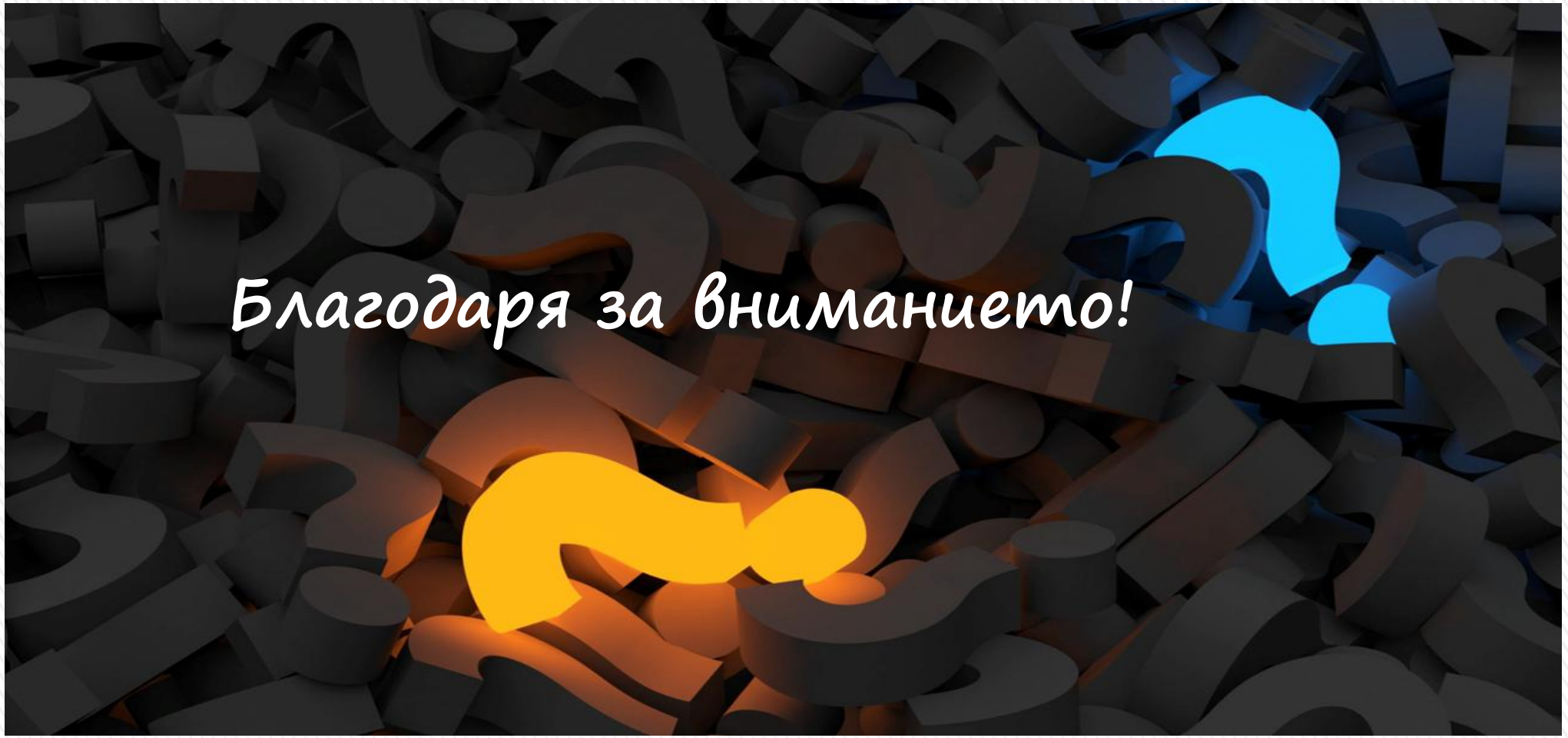
- Някои от най-важните насоки за определяне на добре капсулиран клас:
- Поставяме коментар преди дефиницията на класа, която описва как програмистът трябва да мисли за данните и методите на класа.
  - Ако класът описва например сума пари, програмистът трябва да мисли в долари и центове, а не в това как класът представя парите.
  - Коментарът в този случай трябва да бъде написан, за да помогне на програмиста да мисли по този начин.

Добра практика за капсулиране

- Обявяваме всички променливи на екземпляра в класа като частни.
- Осигуряваме публични методи за достъп за извличане на данните в обект:
  - get методи (getters).
- Също така предоставяме публични методи за всички други основни нужди, които програмистът ще има за манипулиране на данните в класа:
  - set методи (setters).

Добра практика за капсулиране

- Даваме коментар преди всяко име на публичен метод, който напълно уточнява как да се използва метода.
- Всички помощни методи са частни.
- Коментари в дефиницията на класа за да опишем подробности за изпълнението.


Благодаря за вниманието!